Packet-Based Interactive Ray Tracing with CUDA

Jared S. Heinly
Department of Computer Science
Grove City College

Kevin D. Bensema

Christiaan P. Gribble

Figure 1: Our packet-based interactive ray tracing system renders a wide range of scenes of varying geometric complexity at 8-14 frames per second with simple eyelight shading and full texturing using NVIDIA’s Compute Unified Device Architecture.

1 Introduction

As an introduction to high-performance computing in massively parallel environments, this undergraduate work-in-progress explores the implementation of packet-based interactive ray tracing using NVIDIA’s Compute Unified Device Architecture (CUDA).

This poster will present our variation of the BVH-based packet traversal algorithm introduced by Günther et al. [2007]. Currently, our implementation achieves 8-14 frames per second with visibility rays, full texturing, and simple eyelight shading for several scenes of varying geometric complexity.

2 Implementation

We use the packet-based traversal algorithm described by Günther et al. [2007] as a guide in our current implementation. Unlike previous GPU-based ray tracing algorithms [Foley and Sugarman 2005; Popov et al. 2007], this method imposes a packet-based traversal and shares the stack among the hardware threads, effectively amortizing the storage costs over all of the rays in a packet. An implementation of this method requires careful attention to resource utilization, particularly the programmer-managed parallel data cache, or shared memory, of current NVIDIA GPUs.

In the basic packet-based traversal scheme, each ray maps to a thread, and each packet to a SIMD warp. Nodes in the BVH are processed against the packet in turn, and traversal decisions are guided by a majority-vote mechanism using sum reduction.

However, our implementation differs from the original in several significant ways. First, we store three nodes within the shared memory region: the current node and its two children. We observe a small but measurable performance impact if processing the current node does not require access to the device memory, and as a result, we shift the child to be traversed from its current position within the shared memory to the first position before continuing to the next traversal step. Second, we find that packets of 8 × 8 rays, which are processed as two 32-wide SIMD warps and impose some additional synchronization, yield significantly better performance for all test scenes on the target architecture (see below) when compared to a single packet consisting of 32 rays. Finally, our system supports sphere primitives directly and utilizes a relatively naive ray-triangle intersection algorithm, both of which may contribute to the performance issues we discuss below.

3 Discussion

The images generated by our system using several common scenes are depicted in Figure 1. The absolute performance (in frames per second) for these viewpoints—rendered with visibility rays at a resolution of 1024 × 1024 pixels and simple eyelight shading—are obtained using an NVIDIA GeForce 9800 GX2 (compute capability 1.1), and the results for various packet configurations (4 × 8, 8 × 4, 8 × 8, 8 × 12, and 12 × 8) are given in Figure 2. As can be seen, the system supports a wide range of scenes with full texturing while maintaining 8-14 frames per second.

We readily observe that, even without secondary illumination effects such as shadows, our implementation currently fails to achieve the performance reported by Günther et al. [2007] for the conference and fairy scenes, despite using newer hardware to gather the results. The issue is likely a result of resource usage: with 64 rays/packet, our implementation consumes 1200 bytes of shared memory, requires 39 registers, spills 56 bytes to the so-called local memory, and achieves only 25% occupancy. The available registers/multiprocessor is the current limiting factor in our implementation. As occupancy is a key metric for multiprocessor utilization in current NVIDIA GPUs, our system likely suffers such poor performance as a result. If the proposed poster is accepted, we hope to actively discuss this and other potential limitations in our system with GPU computing experts throughout the conference. In addition, we hope to improve our system significantly in the months before the conference, particularly by aggressive reuse of temporary variables to reduce register pressure.

Despite these shortcomings, we have as a first step implemented an interactive ray tracing system using CUDA, and in the process, have learned a great deal about the opportunities and challenges with which programmers must contend to effectively utilize massively parallel computational environments.
Acknowledgments

Parts of this work were funded by the Swezey Scientific Instrumentation and Research Fund. The GPUs used in this research were generously donated by the NVIDIA Corporation through their Professor Partnership Program. The authors thank Johannes Günther and Stefan Popov for their help in the initial stages of this project.

References

